INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

4A STI

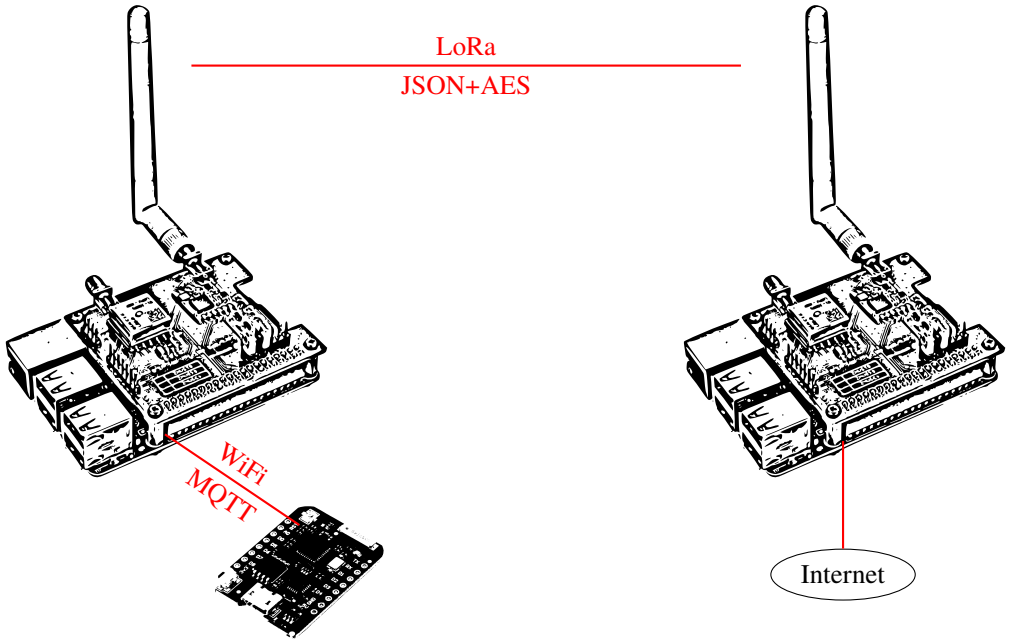
Réseaux non filaires

Projet

IoT, LoRa, WiFi, MQTT, SSL, Mongoose OS, Raspberry Pi & ESP8266

Présentation du projet

Raspberry Pi 3 + LoRa Hat Dragino *Raspberry Pi 3 + LoRa Hat Dragino*



Wemos ESP8266

Le **but du projet** est de réaliser :

- un réseau de capteurs **connectés par WiFi** vers un concentrateur :
  - ils utilisent le **protocole MQTT** pour remonter des mesures vers le concentrateur au travers d'une connexion WiFi ;
  - chaque capteur correspond à un **ESP8266** intégré dans la carte de développement « *Wemos* » ;
  - un **Raspberry Pi** joue le rôle du concentrateur :
    - il exécute un « *broker* » MQTT : *logiciel mosquitto* ;
    - il sert de point d'accès WiFi : *logiciel hostapd* ;
- le framework de développement est **Mongoose OS** permettant de :
  - programmer le système embarqué ESP8266 ;
  - disposer d'une implémentation de TLS ;
- le concentrateur est relié vers une passerelle par l'utilisation des communications **LoRa** :
  - chaque Raspberry Pi est « *coiffé* » d'un dragino, intégrant un « *transceiver* » LoRa ainsi qu'un GPS ;
  - la communication d'une mesure est réalisée au travers de LoRa vers le Raspberry Pi connecté à Internet.

Raspberry Pi & WiFi

Vous installerez et configurerez sur le Raspberry Pi :

- pour la connexion WiFi des ESP8266 :
  - le point d'accès logiciel « `hostapd` » ;
    `https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/`
  - le serveur DHCP permettant de prendre en charge les clients WiFi connectés ;
- le serveur « `mosquitto` » et la sécurité.
  `http://rockingdlabs.dunmire.org/exercises-experiments/ssl-client-certs-to-secure-mqtt`

Communications et sécurité

1. L'ESP8266 va se comporter comme un client MQTT en se connectant au serveur MQTT du Raspberry Pi jouant le rôle de concentrateur :
   - il va transmettre à intervalle régulier une mesure, en tant que « *publisher* » ;
     Ici, la valeur de son ADC, «Analog Digital Converter».
   - sur le « *topic* » « `ESP/ADC` » ;
     Vous pourrez contrôler que cela fonctionne en vous connectant sur le serveur MQTT en tant que «subscriber» :

     xterm
     ```
     $ mosquitto_sub -h 10.20.30.1 -t 'ESP/ADC'
     {ValeurADC: 10}
     ```
   - le Raspberry Pi pourra récupérer les mesures une par une, par exemple en sortie de la commande « `mosquitto_sub` » ;
   - **la connexion entre l'ESP8266 pourra être faire en TLS avec authentification du serveur et, éventuellement, du client par certificat/clé ECC**.

2. le Raspberry Pi concentrateur va transmettre la valeur par LoRa vers le Raspberry Pi assurant la passerelle Internet :
   - la valeur sera **chiffrée par AES avec une clé partagée** entre les deux Raspberry Pis.
     Vous utiliserez les programmes « `rf95_client` » et « `rf95_server` » de démonstration du LoRa, pour les étendre et permettre le choix du message à envoyer correspondant à la valeur récupérée depuis MQTT
   - la valeur récupérée sera simplement affichée sur le Raspberry Pi recevant le message LoRa.
     On pourrait imaginer de traiter les valeurs et permettre leur consultation sous forme d'un serveur Web ou bien en les transmettant sur un autre serveur MQTT situé sur Internet.

Chiffrement ECC : clés et certificats

Pour la génération de la clé et du certificat pour l'AC :

xterm
```
$ openssl ecparam -out ecc.ca.key.pem -name prime256v1 -genkey

$ openssl req -config <(printf
"[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:TRUE") -new -nodes
-subj "/C=FR/L=Bourges/O=RnF/OU=IOT/CN=ACINSA" -x509 -extensions ext -sha256 -key
ecc.ca.key.pem -text -out ecc.ca.cert.pem
```

Pour le serveur MQTT avec son adresse IP (pour un FQDN, il faut un serveur DNS) :

xterm
```
$ openssl ecparam -out ecc.server.key.pem -name prime256v1 -genkey
$ openssl req -config <(printf
"[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:FALSE") -new -subj
"/C=FR/L=Bourges/O=RnF/OU=IOT/CN=10.90.90.254" -reqexts ext -sha256 -key
ecc.server.key.pem -text -out ecc.server.csr.pem

$  openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.key.pem
-CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in
ecc.server.csr.pem -text  -out ecc.server.pem
```

Pour le transfert du certificat sur l'ESP8266 :

xterm
```
$ mos put ecc.ca.cert.pem
```

Si on veut aussi authentifier le client auprès du serveur :

xterm
```
$ openssl ecparam -out ecc.key.pem -name prime256v1 -genkey

$ openssl req -config <(printf
"[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:FALSE") -new -subj
"/C=FR/L=Bourges/O=RnF/OU=IOT/CN=esp8266" -reqexts ext -sha256 -key ecc.key.pem
-text -out ecc.csr.pem

$ openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.key.pem
-CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in
ecc.csr.pem -text  -out ecc.esp8266.pem
```

Pour la configuration du serveur MQTT sur le Raspberry Pi pour TLS :

xterm

```
pef@cube:/etc/mosquitto/conf.d$ cat tls.conf
listener 8883
cafile /home/pef/ECC_CERTIFICATES/ecc.ca.cert.pem
certfile /home/pef/ECC_CERTIFICATES/ecc.server.pem
keyfile /home/pef/ECC_CERTIFICATES/ecc.server.key.pem
```

Mongoose OS : MQTT client et publication sécurisée par ECC

Vous utiliserez la configuration en « *expert view* » :

```
"mqtt": {
  "enable": true,
  "server": "10.90.90.254:8883",
  "client_id": "",
  "user": "",
  "pass": "",
  "reconnect_timeout_min": 2,
  "reconnect_timeout_max": 60,
  "ssl_cert": "",
  "ssl_key": "",
  "ssl_ca_cert": "ecc.ca.cert.pem",
  "ssl_cipher_suites": "",
  "ssl_psk_identity": "",
  "ssl_psk_key": "",
  "clean_session": true,
  "keep_alive": 60,
  "will_topic": "",
  "will_message": ""
},
```

Le code à mettre à la fin du fichier « *init.js* » :

```
load("api_adc.js");
ADC.enable(0);

function envoi() {
  let message = JSON.stringify({ValeurADC: ADC.read(0)});
  let ok = MQTT.pub('ESP/ADC', message, 1);
  print('Published:', ok, '->', message);
}
Timer.set(2000,true,envoi,null);

// Monitor network connectivity.
Net.setStatusEventHandler(function(ev, arg) {
  let evs = '???';
  if (ev === Net.STATUS_DISCONNECTED) {
    evs = 'DISCONNECTED';
  } else if (ev === Net.STATUS_CONNECTING) {
    evs = 'CONNECTING';
  } else if (ev === Net.STATUS_CONNECTED) {
    evs = 'CONNECTED';
  } else if (ev === Net.STATUS_GOT_IP) {
    evs = 'GOT_IP';
  }
  print('== Net event:', ev, evs);
}, null);
```

Ce code lit la valeur fournie par l'ADC et le publie sur le topic « ESP/ADC ».

Vous observerez dans les traces transmises par l'ESP8266 :

```
[Mar 20 21:41:02.421] WPA2 ENTERPRISE VERSION: [v2.0] disable
[Mar 20 21:41:02.421] mgos_wifi_setup_sta  WiFi STA: Connecting to yopa
[Mar 20 21:41:02.428] mgos_http_server_ini HTTP server started on [80]
[Mar 20 21:41:02.436] mg_rpc_channel_mqtt  0x3fff1354 esp8266_64C6BA/rpc/#
[Mar 20 21:41:02.442] mg_rpc_channel_uart  0x3fff16e4 UART0
[Mar 20 21:41:02.449] mgos_init            Init done, RAM: 52104 total, 43152 free, 42408 min free
[Mar 20 21:41:03.307] LED GPIO: 2 button GPIO: 0
[Mar 20 21:41:03.330] mongoose_poll        New heap free LWM: 29504
[Mar 20 21:41:03.337] mgos_net_on_change_c WiFi STA: connecting
[Mar 20 21:41:03.349] == Net event: 1 CONNECTING
[Mar 20 21:41:05.562] scandone
[Mar 20 21:41:06.445] state: 0 -> 2 (b0)
[Mar 20 21:41:06.455] state: 2 -> 3 (0)
```

```
[Mar 20 21:41:06.459] state: 3 -> 5 (10)
[Mar 20 21:41:06.459] add 0
[Mar 20 21:41:06.459] aid 1
[Mar 20 21:41:06.459] cnt
[Mar 20 21:41:07.478]
[Mar 20 21:41:07.478] connected with yopa, channel 1
[Mar 20 21:41:07.478] dhcp client start...
[Mar 20 21:41:07.478] mgos_net_on_change_c WiFi STA: connected
[Mar 20 21:41:07.491] == Net event: 2 CONNECTED
[Mar 20 21:41:08.457] ip:10.90.90.116,mask:255.255.255.0,gw:10.90.90.254
[Mar 20 21:41:08.457] mgos_net_on_change_c WiFi STA: ready, IP 10.90.90.116,
GW 10.90.90.254, DNS 8.8.8.8
[Mar 20 21:41:08.472] == Net event: 3 GOT_IP
[Mar 20 21:41:08.478] mgos_mqtt_global_con MQTT connecting to 10.90.90.254:8883
[Mar 20 21:41:08.538] SW ECDSA verify curve 3 hash_len 32 sig_len 71
[Mar 20 21:41:13.352] SW ECDSA verify curve 3 hash_len 64 sig_len 71
[Mar 20 21:41:17.955] SW ECDH
[Mar 20 21:41:22.327] mongoose_poll        New heap free LWM: 22104
[Mar 20 21:41:22.398] pm open,type:2 0
[Mar 20 21:41:22.415] mgos_mqtt_ev         MQTT Connect (1)
[Mar 20 21:41:22.433] mgos_sntp_query      SNTP query to pool.ntp.org
[Mar 20 21:41:22.442] mgos_mqtt_ev         MQTT CONNACK 0
[Mar 20 21:41:22.448] do_subscribe         Subscribing to 'esp8266_64C6BA/rpc'
[Mar 20 21:41:22.455] do_subscribe         Subscribing to 'esp8266_64C6BA/rpc/#'
```

Chaque segment TCP transmis par l'ESP8266 est chiffré par TLS :

```
pef@cube:~$ sudo tcpdump -i wlxa0f3c11449ec -lnvvX
tcpdump: listening on wlxa0f3c11449ec, link-type EN10MB (Ethernet), capture
size 262144 bytes
22:11:11.857349 IP (tos 0x0, ttl 128, id 207, offset 0, flags [none], proto
TCP (6), length 99)
    10.90.90.116.23290 > 10.90.90.254.8883: Flags [P.], cksum 0x0d9c (cor
rect), seq 12594:12653, ack 782404673, win 5807, length 59
   0x0000:  4500 0063 00cf 0000 8006 6fa0 0a5a 5a74  E..c......o..ZZt
   0x0010:  0a5a 5afe 5afa 22b3 0000 3132 2ea2 8c41  .ZZ.Z."...12...A
   0x0020:  5018 16af 0d9c 0000 1703 0300 3600 0000  P...........6...
   0x0030:  0000 0000 647b 2af8 7c61 d407 bb06 3c42  ....d{*.|a....<B
   0x0040:  287d f6ca d7d3 acd2 c82e 6d08 c443 029b  (}........m..C..
   0x0050:  305b a879 6354 501f 37b5 3a9f 7d4c 02f0  0[.ycTP.7.:.}L..
   0x0060:  c655 4bfa 2016 f529 2440 a2              .UK....)$@.
```

Utilisation de LoRa sur le Raspberry Pi

Le composant LoRa est supporté par un « *hat* » : le dragino.

`http://wiki.dragino.com/index.php?title=Lora/GPS_HAT`.

Le Raspberry Pi et le composant LoRa vont communiquer par l'intermédiaire du bus SPI.

ATTENTION

Un antenne doit **toujours** être connectée sur le port « LoRa », sous peine **d'endommager** le composant LoRa.

Configuration initiale du Raspberry Pi

La procédure de configuration du Raspberry Pi peut changer suivant la version de votre distribution Raspbian.

pi@raspberry

```
$ sudo raspi-config
```

Sélectionner l'option nº7 « *Advanced Options* » :

pi@raspberry

```
 Raspberry Pi Software Configuration Tool (raspi-config)

    1 Change User Password Change password for the current user
    2 Hostname             Set the visible name for this Pi on a network
    3 Boot Options         Configure options for start-up
    4 Localisation Options Set up language and regional settings to match you
    5 Interfacing Options  Configure connections to peripherals
    6 Overclock            Configure overclocking for your Pi
    7 Advanced Options     Configure advanced settings
    8 Update               Update this tool to the latest version
    9 About raspi-config   Information about this configuration tool

                    <Select>                    <Finish>
```

Sélectionner « *A1 Expand Filesystem* » :

pi@raspberry

```
 Raspberry Pi Software Configuration Tool (raspi-config)

  A1 Expand Filesystem Ensures that all of the SD card storage is available to
  A2 Overscan          You may need to configure overscan if black bars are pr
  A3 Memory Split      Change the amount of memory made available to the GPU
  A4 Audio             Force audio out through HDMI or 3.5mm jack
  A5 Resolution        Set a specific screen resolution
  A6 GL Driver         Enable/Disable experimental desktop GL driver

                    <Select>                    <Back>
```

Enfin l'option 5 « *Interfacing Options* » puis l'option « *P4 SPI* » et « *Yes* » :

pi@raspberry

```
 Raspberry Pi Software Configuration Tool (raspi-config)

    P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
    P2 SSH         Enable/Disable remote command line access to your Pi using
    P3 VNC         Enable/Disable graphical remote access to your Pi using Rea
    P4 SPI         Enable/Disable automatic loading of SPI kernel module
    P5 I2C         Enable/Disable automatic loading of I2C kernel module
    P6 Serial      Enable/Disable shell and kernel messages on the serial conn
    P7 1-Wire      Enable/Disable one-wire interface
    P8 Remote GPIO Enable/Disable remote access to GPIO pins

               <Select>                                   <Back>
```

puis l'option « *A1 SPI* » puis « *enable SPI* » et « *load kernel module by default* », puis « *Finish* » et « *Reboot* » :

Pour **connecter le Raspberry Pi par WiFi**, vous éditerez le contenu du fichier « `/etc/network/interfaces` » :

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet dhcp
        wpa-ssid "ssid"
        wpa-psk "password"
```

Pour **mettre à jour le Raspberry Pi**, une fois celui connecté au WiFi et que vous l'aurez vérifié/configuré pour utiliser la route par défaut passant par le WiFi :

pi@raspberry

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo rpi-update
$ sudo reboot
```

Pour **l'utilisation des broches GPIOs** et du bus SPI vous aurez besoin de la bibliothèque bcm2835 :

`http://www.airspayce.com/mikem/bcm2835/`

pi@raspberry

```
$ wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.58.tar.gz
$ tar zxvf bcm2835-1.58.tar.gz
$ cd bcm2835-1.58
$ ./configure
$ make
$ sudo make check
$ sudo make install
```

Pour le **bon fonctionnement** de la bibliothèque bcm2835 sur Raspberry Pi 3, il faut éditer le fichier « `/boot/config.txt` » et ajouter à la fin :

```
dtoverlay=gpio-no-irq
```

Pour **l'utilisation du LoRa**, on utilisera la bilbliothèque suivante :

pi@raspberry
```
$ git clone https://github.com/hallard/RadioHead
```

Vous irez dans le répertoire suivant de la bibliothèque :

pi@raspberry
```
$ cd RadioHead/examples/raspi/rf95
```

Vous modifierez les deux fichier sources : « `rf95_server.cpp` » et « `rf95_client.cpp` », pour sélectionner le dragino :

```
// LoRasPi board
// see https://github.com/hallard/LoRasPI
#define BOARD_LORASPI
```

```
// LoRasPi board
// see https://github.com/hallard/LoRas
PI
//#define BOARD_LORASPI
```

On désactive ce `#define` en premier. ⇒

```
// Dragino Raspberry PI hat
// see https://github.com/dragino/Lora
//#define BOARD_DRAGINO_PIHAT
```

```
// Dragino Raspberry PI hat
// see https://github.com/dragino/Lora
#define BOARD_DRAGINO_PIHAT
```

Puis vous compilerez et lancerez une des versions du logiciel :

pi@raspberry
```
$ make
$ sudo ./rf95_client
```

Vous aurez maintenant deux Raspberry Pi communiquant par LoRa.

Terminaison de la plateforme Capteurs/Passerelle LoRa

Il ne restera plus qu'à :

- ▷ **récupérer le message** envoyée à l'aide de MQTT du ou des ESP8266 sur un Raspberry Pi ;
- ▷ **transmettre le message par LoRa** depuis ce Raspberry pi
- ▷ **recevoir par LoRa le message** sur l'autre Raspberry Pi ;

On peut chiffrer le message transmis par LoRa en utilisant AES et une clé préinstallée et partagée sur les deux Raspberry Pi.